Client Side Classes

09872502.060401

Class HTTPJDBCCallableStatement

```
package com.VSDV.HTTPJDBC.Client;

import java.sql.*;
import java.util.*;
import java.math.*;

import com.VSDV.HTTPJDBC.Common.*;

/**
 * LFRT class implementing java.sql.CallableStatement
 */
public class HTTPJDBCCallableStatement extends HTTPJDBCPreparedStatement
implements CallableStatement {

  public HTTPJDBCCallableStatement(HTTPJDBCConnection con, String objectName,
Object[] parameters) {
    super(con, parameters);
  }

  public void initializeStatement(HTTPJDBCConnection con, Object[] parameters){
    InObject defaultMethodObject = new InObject(con.getName(), "", parameters, true);
    defaultMethodObject.setMethod("prepareCall");
    addToMethodList(defaultMethodObject);
  }

  public void registerOutParameter(int parameterIndex, int sqlType) throws
SQLException {
    InObject inObject = new InObject(objectName, "registerOutParameter", new Object[]
{new Integer(parameterIndex), new Integer(sqlType) }, false );
    addToMethodList(inObject);
  }

  public void registerOutParameter(int parameterIndex, int sqlType, int scale) throws
SQLException {
    InObject inObject = new InObject(objectName, "registerOutParameter", new Object[]
{new Integer(parameterIndex), new Integer(sqlType), new Integer(scale) } ,false);
    addToMethodList(inObject);
  }

  public boolean wasNull() throws SQLException {
    Boolean b = (Boolean)session.call(objectName, "wasNull" , true);
    return b.booleanValue();
  }
  public String getString(int parameterIndex) throws SQLException {
    return (String)session.call(objectName, "getString", new Object[] {new
Integer(parameterIndex) } ,true);
  }

  public boolean getBoolean(int parameterIndex) throws SQLException {
    Boolean b = (Boolean)session.call(objectName, "getBoolean", new Object[] {new
Integer(parameterIndex) }, true );
    return b.booleanValue();
  }

  public byte getByte(int parameterIndex) throws SQLException {
    Byte b = (Byte)session.call(objectName, "getByte", new Object[] {new
Integer(parameterIndex) } , true  );
    return b.byteValue();
  }

  public short getShort(int parameterIndex) throws SQLException {
    Short b = (Short)session.call(objectName, "getShort", new Object[] {new
Integer(parameterIndex) } ,true );
    return b.shortValue();
  }

  public int getInt(int parameterIndex) throws SQLException {
    Integer b = (Integer)session.call(objectName, "getInt", new Object[] {new
Integer(parameterIndex) },true );
    return b.intValue();
  }

  public long getLong(int parameterIndex) throws SQLException {
    Long b = (Long)session.call(objectName, "getLong", new Object[] {new
Integer(parameterIndex) }, true  );
    return b.longValue();
  }

  public float getFloat(int parameterIndex) throws SQLException {
    Float b = (Float)session.call(objectName, "getFloat", new Object[] {new
Integer(parameterIndex) }, true );
    return b.floatValue();
  }

  public double getDouble(int parameterIndex) throws SQLException {
    Double b = (Double)session.call(objectName, "getDouble", new Object[] {new
Integer(parameterIndex) } , true);
    return b.doubleValue();
  }

  public BigDecimal getBigDecimal(int parameterIndex, int scale) throws SQLException
 {
   return (BigDecimal)session.call(objectName, "getBigDecimal", new Object[] {new
Integer(parameterIndex), new Integer(scale) }, true);
  }

  public byte[] getBytes(int parameterIndex) throws SQLException {
   return (byte[])session.call(objectName, "getBytes", new Object[] {new
Integer(parameterIndex) } , true);
  }

  public java.sql.Date getDate(int parameterIndex) throws SQLException {
   return (java.sql.Date)session.call(objectName, "getDate", new Object[] {new
Integer(parameterIndex) }, true );
  }

  public java.sql.Time getTime(int parameterIndex) throws SQLException {
   return (java.sql.Time)session.call(objectName, "getTime", new Object[] {new
Integer(parameterIndex) }, true );
  }

  public java.sql.Timestamp getTimestamp(int parameterIndex) throws SQLException {
   return (java.sql.Timestamp)session.call(objectName, "getTimestamp", new Object[]
{new Integer(parameterIndex) } ,true);
  }

  public Object getObject(int parameterIndex) throws SQLException {
   return session.call(objectName, "getObject", new Object[] {new
Integer(parameterIndex) },true );
  }

             //--------------------------JDBC 2.0----------------------------

  public BigDecimal getBigDecimal(int parameterIndex) throws SQLException {
   return (BigDecimal)session.call(objectName, "getBigDecimal", new Object[] {new
Integer(parameterIndex) } ,true);
  }

  public Object  getObject (int i, java.util.Map map) throws SQLException {
   return (Object)session.call(objectName, "getObject", new Object[] {map},true );
  }

  public Ref getRef (int i) throws SQLException {
   return (Ref)session.call(objectName, "getRef", new Object[] {new Integer(i) },true );
  }

  public Blob getBlob (int i) throws SQLException {
    return (Blob)session.call(objectName, "getBlob", new Object[] {new Integer(i) }
,true);
  }

  public Clob getClob (int i) throws SQLException {
    return (Clob)session.call(objectName, "getClob", new Object[] {new Integer(i) }
,true);
  }

  public Array getArray (int i) throws SQLException {
    return (Array)session.call(objectName, "getArray", new Object[] {new Integer(i) },true
);
  }

  public java.sql.Date getDate(int parameterIndex, Calendar cal)
                throws SQLException {
    return (java.sql.Date)session.call(objectName, "getDate", new Object[] {new
Integer(parameterIndex), cal } ,true);
  }

  public java.sql.Time getTime(int parameterIndex, Calendar cal)
                throws SQLException {
    return (java.sql.Time)session.call(objectName, "getTime", new Object[] {new
Integer(parameterIndex), cal },true );
  }

  public java.sql.Timestamp getTimestamp(int parameterIndex, Calendar cal)
                throws SQLException {
    return (java.sql.Timestamp)session.call(objectName, "getTimestamp", new Object[]
{new Integer(parameterIndex), cal } ,true);
  }

  public void registerOutParameter (int paramIndex, int sqlType, String typeName)
throws SQLException {
    InObject inObject = new InObject(objectName, "registerOutParameter", new Object[]
{new Integer(paramIndex), new Integer(sqlType), typeName } ,false);
    addToMethodList(inObject);
  }
}
```

09837250 . 060401

Class HTTPJDBCConnection

```
package com.VSDV.HTTPJDBC.Client;

import java.sql.*;
import java.util.*;

import com.VSDV.HTTPJDBC.Common.*;

/**
 * LFRT class implementing java.sql.Connection
 */
public class HTTPJDBCConnection extends JDBCStub implements Connection {

  HTTPJDBCSession session;
  boolean closed, isReadOnly;

  public HTTPJDBCConnection(HTTPJDBCSession session, String objectName) {
        super(objectName);  // This names the object name
    this.session = session; // Each conneciton is assigned to a HTTPJDBCSESSION
    closed = false;
  }

  public String makeTestCall() throws SQLException {
    return (String)session.call("testobject", "testMethod", true);
  }

  public void clearWarnings() throws SQLException {
    session.call(objectName, "clearWarnings", false);
  }

  public void close() throws SQLException {
        session.call(objectName, "close", false);
    closed = true;
  }

  public void commit() throws SQLException {
    session.call(objectName, "commit", false );
  }

  public boolean getAutoCommit() throws SQLException {
        Boolean b = (Boolean)session.call(objectName, "getAutoCommit", true );
        return b.booleanValue();
  }

  public String getCatalog() throws SQLException {
```

```
  return (String)session.call(objectName, "getCatalog", true);
 }

 public DatabaseMetaData getMetaData() throws SQLException {
  String n = (String)session.call(objectName, "getMetaData", true);
  HTTPJDBCDatabaseMetaData stmt = new HTTPJDBCDatabaseMetaData(this, n);
  return stmt;
 }

 public int getTransactionIsolation() throws SQLException {
  Integer i =  (Integer)session.call(objectName, "getTransactionIsolation", true);
        return i.intValue();
 }

 public SQLWarning getWarnings() throws SQLException {
        return (SQLWarning)session.call(objectName, "getWarnings", true);
 }

 public Statement createStatement() throws SQLException {
  HTTPJDBCStatement stmt = new HTTPJDBCStatement(this);
  return stmt;
 }

 public boolean isClosed() throws SQLException {
        return closed;
 }

 public boolean isReadOnly() throws SQLException {
        return isReadOnly;
 }

 public String nativeSQL(String sql) throws SQLException {
  return (String)session.call(objectName, "nativeSQL", new Object[] {sql}, true );
 }

 public CallableStatement prepareCall(String sql) throws SQLException {
        HTTPJDBCCallableStatement stmt = new HTTPJDBCCallableStatement(this,
null, new Object[] {sql});
  return stmt;
 }

 public PreparedStatement prepareStatement(String sql) throws SQLException {
        HTTPJDBCPreparedStatement stmt = new HTTPJDBCPreparedStatement(this,
new Object[] {sql});
  return stmt;
 }
 public void rollback() throws SQLException {
          session.call(objectName, "rollback", false);
 }

 public void setAutoCommit(boolean autoCommit) throws SQLException {
          session.call(objectName, "setAutoCommit", new Object[] {new
Boolean(autoCommit)}, false );
 }

 public void setCatalog(String catalog) throws SQLException {
          session.call(objectName, "setCatalog", new Object[] {catalog}, false );
 }

 public void setReadOnly(boolean readOnly) throws SQLException {
          session.call(objectName, "setReadOnly", new Object[] {new
Boolean(readOnly)}, false );
 }

 public void setTransactionIsolation(int level) throws SQLException {
          session.call(objectName, "setTransactionIsolation", new Object[] {new
Integer(level)}, false );
 }

 HTTPJDBCSession getHTTPJDBCSession() { return session; }

 public Statement createStatement(int resultSetType, int resultSetConcurrency) throws
SQLException {
   return null;
 }
 public PreparedStatement prepareStatement(String sql, int resultSetType, int
resultSetConcurrency) throws SQLException {
   return null;
 }

 public CallableStatement prepareCall(String sql, int resultSetType, int
resultSetConcurrency) throws SQLException {
   return null;
 }

 public void setTypeMap(Map map) throws SQLException { }

 public Map getTypeMap() throws SQLException {
   return null;
 }
```

```
  public String getSessionID(){
    return session.getID();
  }

  public void finalize(){
    System.out.println("Closing connection");
    try{
      if (!isClosed()){
        close();
      }
    } catch(SQLException sqle){
      sqle.printStackTrace();
    }
  }
}
```

Class HTTPJDBCDatabaseMetadata

```
package com.VSDV.HTTPJDBC.Client;

import java.sql.*;

import com.VSDV.HTTPJDBC.Common.*;

public class HTTPJDBCDatabaseMetaData extends JDBCStub implements
DatabaseMetaData {

 private HTTPJDBCSession session;
 private HTTPJDBCConnection connection;

 public HTTPJDBCDatabaseMetaData(HTTPJDBCConnection connection, String
objectName) {
   super(objectName);
   this.connection = connection;
   session = connection.getHTTPJDBCSession();
 }

        public boolean allProceduresAreCallable() throws SQLException{
        return ((Boolean)session.call(objectName, "allProceduresAreCallable",
true)).booleanValue();
 }

        public boolean allTablesAreSelectable() throws SQLException{
        return ((Boolean)session.call(objectName, "allTablesAreSelectable",
true)).booleanValue();
 }

        public String getURL() throws SQLException{
        return (String)session.call(objectName, "getURL", true);
 }

        public String getUserName() throws SQLException{
        return (String)session.call(objectName, "getUserName", true);
 }

        public boolean isReadOnly() throws SQLException{
        return ((Boolean)session.call(objectName, "getUserName", true)).booleanValue();
 }

        public boolean nullsAreSortedHigh() throws SQLException{
```

09872502.060101

```
        return ((Boolean)session.call(objectName, "nullsAreSortedHigh",
true)).booleanValue();
 }

        public boolean nullsAreSortedLow() throws SQLException{
        return ((Boolean)session.call(objectName, "nullsAreSortedLow",
true)).booleanValue();
 }

        public boolean nullsAreSortedAtStart() throws SQLException{
        return ((Boolean)session.call(objectName, "nullsAreSortedAtStart",
true)).booleanValue();
 }

        public boolean nullsAreSortedAtEnd() throws SQLException{
        return ((Boolean)session.call(objectName, "nullsAreSortedAtEnd",
true)).booleanValue();
 }

        public String getDatabaseProductName() throws SQLException{
        return (String)session.call(objectName, "getDatabaseProductName", true);
 }

        public String getDatabaseProductVersion() throws SQLException{
        return (String)session.call(objectName, "getDatabaseProductVersion", true);
 }

        public String getDriverName() throws SQLException{
        return (String)session.call(objectName, "getDriverName", true);
 }

        public String getDriverVersion() throws SQLException{
        return (String)session.call(objectName, "getDriverVersion", true);
 }

        public int getDriverMajorVersion(){
        return 1;
 }

        public int getDriverMinorVersion(){
        return 0;
 }

        public boolean usesLocalFiles() throws SQLException{
        return ((Boolean)session.call(objectName, "usesLocalFiles",
true)).booleanValue();
```

```
 }

        public boolean usesLocalFilePerTable() throws SQLException{
        return ((Boolean)session.call(objectName, "usesLocalFilePerTable",
true)).booleanValue();
 }

        public boolean supportsMixedCaseIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsMixedCaseIdentifiers",
true)).booleanValue();
 }

        public boolean storesUpperCaseIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName, "storesUpperCaseIdentifiers",
true)).booleanValue();
 }

        public boolean storesLowerCaseIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName, "storesLowerCaseIdentifiers",
true)).booleanValue();
 }

        public boolean storesMixedCaseIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName, "storesMixedCaseIdentifiers",
true)).booleanValue();
 }

        public boolean supportsMixedCaseQuotedIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsMixedCaseQuotedIdentifiers", true)).booleanValue();
 }

        public boolean storesUpperCaseQuotedIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName, "storesLowerCaseIdentifiers",
true)).booleanValue();
 }

        public boolean storesLowerCaseQuotedIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName, "storesUpperCaseQuotedIdentifiers",
true)).booleanValue();
 }

        public boolean storesMixedCaseQuotedIdentifiers() throws SQLException{
        return ((Boolean)session.call(objectName, "storesMixedCaseQuotedIdentifiers",
true)).booleanValue();
 }
```

```
        public String getIdentifierQuoteString() throws SQLException{
        return (String)session.call(objectName, "getIdentifierQuoteString", true);
    }

        public String getSQLKeywords() throws SQLException{
        return (String)session.call(objectName, "getSQLKeywords", true);
    }

        public String getNumericFunctions() throws SQLException{
        return (String)session.call(objectName, "getNumericFunctions", true);
    }

        public String getStringFunctions() throws SQLException{
        return (String)session.call(objectName, "getStringFunctions", true);
    }

        public String getSystemFunctions() throws SQLException{
        return (String)session.call(objectName, "getSystemFunctions", true);
    }

        public String getTimeDateFunctions() throws SQLException{
        return (String)session.call(objectName, "getTimeDateFunctions", true);
    }

        public String getSearchStringEscape() throws SQLException{
        return (String)session.call(objectName, "getSearchStringEscape", true);
    }

        public String getExtraNameCharacters() throws SQLException{
        return (String)session.call(objectName, "getExtraNameCharacters", true);
    }

        public boolean supportsAlterTableWithAddColumn() throws SQLException{
        return ((Boolean)session.call(objectName, "storesLowerCaseIdentifiers",
true)).booleanValue();
    }

        public boolean supportsAlterTableWithDropColumn() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsAlterTableWithAddColumn",
true)).booleanValue();
    }

        public boolean supportsColumnAliasing() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsColumnAliasing",
true)).booleanValue();
```

```
}

        public boolean nullPlusNonNullIsNull() throws SQLException{
        return ((Boolean)session.call(objectName, "nullPlusNonNullIsNull",
true)).booleanValue();
 }

        public boolean supportsConvert() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsConvert",
true)).booleanValue();
 }

        public boolean supportsConvert(int fromType, int toType) throws
SQLException{
        return ((Boolean)session.call(objectName, "supportsConvert",
true)).booleanValue();
 }

        public boolean supportsTableCorrelationNames() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsTableCorrelationNames",
true)).booleanValue();
 }

        public boolean supportsDifferentTableCorrelationNames() throws
SQLException{
        return ((Boolean)session.call(objectName,
"supportsDifferentTableCorrelationNames", true)).booleanValue();
 }

        public boolean supportsExpressionsInOrderBy() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsExpressionsInOrderBy",
true)).booleanValue();
 }

        public boolean supportsOrderByUnrelated() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsOrderByUnrelated",
true)).booleanValue();
 }

        public boolean supportsGroupBy() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsGroupBy",
true)).booleanValue();
 }

        public boolean supportsGroupByUnrelated() throws SQLException{
```

```
        return ((Boolean)session.call(objectName, "supportsGroupByUnrelated",
true)).booleanValue();
    }

        public boolean supportsGroupByBeyondSelect() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsGroupByBeyondSelect",
true)).booleanValue();
    }

        public boolean supportsLikeEscapeClause() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsLikeEscapeClause",
true)).booleanValue();
    }

        public boolean supportsMultipleResultSets() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsMultipleResultSets",
true)).booleanValue();
    }

        public boolean supportsMultipleTransactions() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsMultipleTransactions",
true)).booleanValue();
    }

        public boolean supportsNonNullableColumns() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsNonNullableColumns",
true)).booleanValue();
    }

        public boolean supportsMinimumSQLGrammar() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsMinimumSQLGrammar",
true)).booleanValue();
    }

        public boolean supportsCoreSQLGrammar() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsCoreSQLGrammar",
true)).booleanValue();
    }

        public boolean supportsExtendedSQLGrammar() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsExtendedSQLGrammar",
true)).booleanValue();
    }

        public boolean supportsANSI92EntryLevelSQL() throws SQLException{
```

```
        return ((Boolean)session.call(objectName, "supportsANSI92EntryLevelSQL",
true)).booleanValue();
  }

        public boolean supportsANSI92IntermediateSQL() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsANSI92IntermediateSQL",
true)).booleanValue();
  }

        public boolean supportsANSI92FullSQL() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsANSI92FullSQL",
true)).booleanValue();
  }


        public boolean supportsIntegrityEnhancementFacility() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsIntegrityEnhancementFacility", true)).booleanValue();
  }


        public boolean supportsOuterJoins() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsOuterJoins",
true)).booleanValue();
  }


        public boolean supportsFullOuterJoins() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsFullOuterJoins",
true)).booleanValue();
  }


        public boolean supportsLimitedOuterJoins() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsLimitedOuterJoins",
true)).booleanValue();
  }


        public String getSchemaTerm() throws SQLException{
        return (String)session.call(objectName, "getSchemaTerm", true);
  }


        public String getProcedureTerm() throws SQLException{
        return (String)session.call(objectName, "getProcedureTerm", true);
  }

        public String getCatalogTerm() throws SQLException{
        return (String)session.call(objectName, "getCatalogTerm", true);
  }

        public boolean isCatalogAtStart() throws SQLException{
        return ((Boolean)session.call(objectName, "isCatalogAtStart",
true)).booleanValue();
  }

        public String getCatalogSeparator() throws SQLException{
        return (String)session.call(objectName, "getCatalogSeparator", true);
  }

        public boolean supportsSchemasInDataManipulation() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsSchemasInDataManipulation", true)).booleanValue();
  }


        public boolean supportsSchemasInProcedureCalls() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsSchemasInProcedureCalls",
true)).booleanValue();
  }


        public boolean supportsSchemasInTableDefinitions() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsSchemasInTableDefinitions",
true)).booleanValue();
  }


        public boolean supportsSchemasInIndexDefinitions() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsSchemasInIndexDefinitions",
true)).booleanValue();
  }


        public boolean supportsSchemasInPrivilegeDefinitions() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsSchemasInPrivilegeDefinitions", true)).booleanValue();
  }


        public boolean supportsCatalogsInDataManipulation() throws SQLException{
```

```
        return ((Boolean)session.call(objectName,
"supportsCatalogsInDataManipulation", true)).booleanValue();
 }


        public boolean supportsCatalogsInProcedureCalls() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsCatalogsInProcedureCalls",
true)).booleanValue();
 }


        public boolean supportsCatalogsInTableDefinitions() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsCatalogsInTableDefinitions",
true)).booleanValue();
 }


        public boolean supportsCatalogsInIndexDefinitions() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsCatalogsInIndexDefinitions",
true)).booleanValue();
 }


        public boolean supportsCatalogsInPrivilegeDefinitions() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsCatalogsInPrivilegeDefinitions", true)).booleanValue();
 }


        public boolean supportsPositionedDelete() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsPositionedDelete",
true)).booleanValue();
 }


        public boolean supportsPositionedUpdate() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsPositionedUpdate",
true)).booleanValue();
 }


        public boolean supportsSelectForUpdate() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsSelectForUpdate",
true)).booleanValue();
 }
```

```
        public boolean supportsStoredProcedures() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsStoredProcedures",
true)).booleanValue();
  }


        public boolean supportsSubqueriesInComparisons() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsSubqueriesInComparisons",true)).booleanValue();
  }


        public boolean supportsSubqueriesInExists() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsSubqueriesInExists",
true)).booleanValue();
  }


        public boolean supportsSubqueriesInIns() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsSubqueriesInIns",
true)).booleanValue();
  }


        public boolean supportsSubqueriesInQuantifieds() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsSubqueriesInQuantifieds",
true)).booleanValue();
  }

        public boolean supportsCorrelatedSubqueries() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsCorrelatedSubqueries",
true)).booleanValue();
  }

        public boolean supportsUnion() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsUnion",
true)).booleanValue();
  }

        public boolean supportsUnionAll() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsUnionAll",
true)).booleanValue();
  }

        public boolean supportsOpenCursorsAcrossCommit() throws SQLException{
```

09872502.060101

```
        return ((Boolean)session.call(objectName,
"supportsOpenCursorsAcrossCommit", true)).booleanValue();
  }

        public boolean supportsOpenCursorsAcrossRollback() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsOpenCursorsAcrossRollback", true)).booleanValue();
  }

        public boolean supportsOpenStatementsAcrossCommit() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsOpenStatementsAcrossCommit", true)).booleanValue();
  }

        public boolean supportsOpenStatementsAcrossRollback() throws SQLException{
        return ((Boolean)session.call(objectName,
"supportsOpenStatementsAcrossRollback", true)).booleanValue();
  }

        public int getMaxBinaryLiteralLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxBinaryLiteralLength",
true)).intValue();
  }

        public int getMaxCharLiteralLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxCharLiteralLength",
true)).intValue();
  }

        public int getMaxColumnNameLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxColumnNameLength",
true)).intValue();
  }

        public int getMaxColumnsInGroupBy() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxColumnsInGroupBy",
true)).intValue();
  }

        public int getMaxColumnsInIndex() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxColumnsInIndex",
true)).intValue();
  }

        public int getMaxColumnsInOrderBy() throws SQLException{
```

```
        return ((Integer)session.call(objectName, "getMaxColumnsInOrderBy",
true)).intValue();
  }

        public int getMaxColumnsInSelect() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxColumnsInSelect",
true)).intValue();
  }

        public int getMaxColumnsInTable() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxColumnsInTable",
true)).intValue();
  }

        public int getMaxConnections() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxConnections", true)).intValue();
  }

        public int getMaxCursorNameLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxCursorNameLength",
true)).intValue();
  }

        public int getMaxIndexLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxIndexLength",
true)).intValue();
  }

        public int getMaxSchemaNameLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxSchemaNameLength",
true)).intValue();
  }

        public int getMaxProcedureNameLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxProcedureNameLength",
true)).intValue();
  }

        public int getMaxCatalogNameLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxCatalogNameLength",
true)).intValue();
  }

        public int getMaxRowSize() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxRowSize", true)).intValue();
  }
```

```
        public boolean doesMaxRowSizeIncludeBlobs() throws SQLException{
        return ((Boolean)session.call(objectName, "doesMaxRowSizeIncludeBlobs",
true)).booleanValue();
 }

        public int getMaxStatementLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxStatementLength",
true)).intValue();
 }

        public int getMaxStatements() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxStatementLength",
true)).intValue();
 }

        public int getMaxTableNameLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxStatementLength",
true)).intValue();
 }

        public int getMaxTablesInSelect() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxStatementLength",
true)).intValue();
 }

        public int getMaxUserNameLength() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxStatementLength",
true)).intValue();
 }

        public int getDefaultTransactionIsolation() throws SQLException{
        return ((Integer)session.call(objectName, "getMaxStatementLength",
true)).intValue();
 }

        public boolean supportsTransactions() throws SQLException{
        return ((Boolean)session.call(objectName, "getMaxStatementLength",
true)).booleanValue();
 }

        public boolean supportsTransactionIsolationLevel(int level) throws
SQLException{
        return ((Boolean)session.call(objectName, "getMaxStatementLength",
true)).booleanValue();
 }
```

```
        public boolean supportsDataDefinitionAndDataManipulationTransactions()
throws SQLException{
        return ((Boolean)session.call(objectName, "getMaxStatementLength",
true)).booleanValue();
  }

        public boolean supportsDataManipulationTransactionsOnly() throws
SQLException{
        return ((Boolean)session.call(objectName, "getMaxStatementLength",
true)).booleanValue();
  }

        public boolean dataDefinitionCausesTransactionCommit() throws
SQLException{
        return ((Boolean)session.call(objectName, "getMaxStatementLength",
true)).booleanValue();
  }

        public boolean dataDefinitionIgnoredInTransactions() throws SQLException{
        return ((Boolean)session.call(objectName, "getMaxStatementLength",
true)).booleanValue();
  }

        public ResultSet getProcedures(String catalog, String schemaPattern, String
procedureNamePattern) throws SQLException{
        return (ResultSet)session.call(objectName, "getProcedures", new Object[] {
catalog, schemaPattern, procedureNamePattern}, true);
  }

        public ResultSet getProcedureColumns(String catalog, String schemaPattern,
String procedureNamePattern, String columnNamePattern) throws SQLException{
        return (ResultSet)session.call(objectName, "getProcedureColumns", new Object[]
{ catalog, schemaPattern, procedureNamePattern, columnNamePattern}, true);
  }

        public ResultSet getTables(String catalog, String schemaPattern, String
tableNamePattern, String types[]) throws SQLException{
        return (ResultSet)session.call(objectName, "getTables", new Object[] {catalog,
schemaPattern, tableNamePattern, types}, true);
  }

        public ResultSet getSchemas() throws SQLException{
        return (ResultSet)session.call(objectName, "getSchemas", true);
  }
```

```
        public ResultSet getCatalogs() throws SQLException{
        return (ResultSet)session.call(objectName, "getCatalogs", true);
  }

        public ResultSet getTableTypes() throws SQLException{
        return (ResultSet)session.call(objectName, "getTableTypes", true);
  }

        public ResultSet getColumns(String catalog, String schemaPattern, String
tableNamePattern, String columnNamePattern) throws SQLException{
        return (ResultSet)session.call(objectName, "getColumns", new Object[]{catalog,
schemaPattern, tableNamePattern, columnNamePattern}, true);
  }

        public ResultSet getColumnPrivileges(String catalog, String schema,String table,
String columnNamePattern) throws SQLException{
        return (ResultSet)session.call(objectName, "getColumnPrivileges", new Object[]
{catalog, schema, table, columnNamePattern}, true);
  }

        public ResultSet getTablePrivileges(String catalog, String schemaPattern, String
tableNamePattern) throws SQLException{
        return (ResultSet)session.call(objectName, "getTablePrivileges", new Object[]
{catalog, schemaPattern, tableNamePattern}, true);
  }

        public ResultSet getBestRowIdentifier(String catalog, String schema, String table,
int scope, boolean nullable) throws SQLException{
        return (ResultSet)session.call(objectName, "getBestRowIdentifier", new Object[]
{catalog, schema, table, new Integer(scope), new Boolean(nullable)}, true);
  }

        public ResultSet getVersionColumns(String catalog, String schema, String table)
throws SQLException{
        return (ResultSet)session.call(objectName, "getVersionColumns", new Object[]
{catalog, schema, table}, true);
  }

        public ResultSet getPrimaryKeys(String catalog, String schema, String table)
throws SQLException{
        return (ResultSet)session.call(objectName, "getPrimaryKeys", new Object[]
{catalog, schema, table}, true);
  }

        public ResultSet getImportedKeys(String catalog, String schema, String table)
throws SQLException{
        return (ResultSet)session.call(objectName, "getImportedKeys", new Object[]
{catalog, schema, table}, true);
 }

        public ResultSet getExportedKeys(String catalog, String schema, String table)
throws SQLException{
        return (ResultSet)session.call(objectName, "getExportedKeys", new Object[]
{catalog, schema, table}, true);
 }

        public ResultSet getCrossReference( String primaryCatalog, String
primarySchema, String primaryTable, String foreignCatalog, String foreignSchema,
String foreignTable) throws SQLException{
        return (ResultSet)session.call(objectName, "getCrossReference", new Object[]
{primaryCatalog, primarySchema, primaryTable, foreignSchema, foreignTable}, true);
 }

        public ResultSet getTypeInfo() throws SQLException{
        return (ResultSet)session.call(objectName, "getTypeInfo", true);
 }

        public ResultSet getIndexInfo(String catalog, String schema, String table, boolean
unique, boolean approximate) throws SQLException{
        return (ResultSet)session.call(objectName, "getIndexInfo", new Object[]{catalog,
schema, table, new Boolean(unique), new Boolean(approximate)}, true);
 }

  public boolean supportsResultSetType(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "supportsResultSetType", new
Object[]{new Integer(type)}, true)).booleanValue();
 }

  public boolean supportsResultSetConcurrency(int type, int concurrency) throws
SQLException{
        return ((Boolean)session.call(objectName, "supportsResultSetConcurrency", new
Object[]{new Integer(type), new Integer(concurrency)}, true)).booleanValue();
 }

  public boolean ownUpdatesAreVisible(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "ownUpdatesAreVisible", new
Object[]{new Integer(type)}, true)).booleanValue();
 }

  public boolean ownDeletesAreVisible(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "ownDeletesAreVisible", new
Object[]{new Integer(type)}, true)).booleanValue();
```

```
  }

  public boolean ownInsertsAreVisible(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "ownInsertsAreVisible", new
Object[]{new Integer(type)}, true)).booleanValue();
  }

  public boolean othersUpdatesAreVisible(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "othersUpdatesAreVisible", new
Object[]{new Integer(type)}, true)).booleanValue();
  }

  public boolean othersDeletesAreVisible(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "othersDeletesAreVisible", new
Object[]{new Integer(type)}, true)).booleanValue();
  }

  public boolean othersInsertsAreVisible(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "othersInsertsAreVisible", new
Object[]{new Integer(type)}, true)).booleanValue();
  }

  public boolean updatesAreDetected(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "updatesAreDetected", new
Object[]{new Integer(type)}, true)).booleanValue();
  }

  public boolean deletesAreDetected(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "deletesAreDetected", new
Object[]{new Integer(type)}, true)).booleanValue();
  }

  public boolean insertsAreDetected(int type) throws SQLException{
        return ((Boolean)session.call(objectName, "insertsAreDetected", new
Object[]{new Integer(type)}, true)).booleanValue();
  }

  public boolean supportsBatchUpdates() throws SQLException{
        return ((Boolean)session.call(objectName, "supportsBatchUpdates",
true)).booleanValue();
  }

  public ResultSet getUDTs(String catalog, String schemaPattern, String
typeNamePattern, int[] types) throws SQLException{
        return (ResultSet)session.call(objectName, "getUDTs", new Object[]{ catalog,
schemaPattern, typeNamePattern, types}, true);
```

```
 }

 public Connection getConnection() throws SQLException{
         return connection;
 }
}
```

Class HTTPJDBCDriver

```
package com.VSDV.HTTPJDBC.Client;

import java.sql.*;
import java.util.*;

import com.VSDV.HTTPJDBC.Common.*;
import com.VSDV.Global.*;

/**
 * Generic JDBC driver for tunneling all JDBC call over HTTP
 */
public class HTTPJDBCDriver implements Driver {

  private static HTTPJDBCDriver driver = new HTTPJDBCDriver();
  private HTTPJDBCSession session;

  public HTTPJDBCDriver() {
   try {
   Globals globals = Globals.getInstance();
   session = new HTTPJDBCSession(globals.getWebHost(), globals.getHttpPort(), false);
   DriverManager.registerDriver(this);
        } catch (Exception ex) { ex.printStackTrace(); }
  }

  /**
   * Connects to servlet creates session
   * servlet in turn creates real JDBC connection
   * If there is a problem on the servlet side we wrap
   * it's exception in an SQLException
   */
  public Connection connect(String url,
                    Properties info)
                throws SQLException {
        StringTokenizer tokenizer = new StringTokenizer(url, ":");
        String protocol = tokenizer.nextToken();
        String host = tokenizer.nextToken();
        int port = Integer.parseInt(tokenizer.nextToken() );
        boolean secure = false;
        if (protocol.equals("httpjdbc") )
          secure = false;
        else if (protocol.equals("httpsjdbc") ) {
    secure = true;
        }
  String objectName = (String)session.call("driver", "connect", new Object[] {url, info},
true );
        HTTPJDBCConnection con = new HTTPJDBCConnection(session, objectName);
         return con;
 }

 public boolean acceptsURL(String url) throws SQLException {
  StringTokenizer tokenizer = new StringTokenizer(url, ":");
         String a = tokenizer.nextToken();
         if (a.equals("httpjdbc") )
          return true;
         else if (a.equals("httpsjdbc") )
          return true;
         else
          return false;
 }

 public DriverPropertyInfo[] getPropertyInfo(String url,
                              Properties info) throws SQLException {
         return null;
 }

 public int getMajorVersion() { return 1; }

 public int getMinorVersion() { return 1; }

 public boolean jdbcCompliant() { return true; }
}
```

Class HTTPJDBCPreparedStatement

```
package com.VSDV.HTTPJDBC.Client;

import java.sql.*;
import java.math.*;
import java.io.*;
import java.util.*;

import com.VSDV.HTTPJDBC.Common.*;

public class HTTPJDBCPreparedStatement extends HTTPJDBCStatement implements
PreparedStatement {

  public HTTPJDBCPreparedStatement(HTTPJDBCConnection con, Object[]
parameters) {
    super(con, parameters);
  }

  public void initializeStatement(HTTPJDBCConnection con, Object[] parameters){
    InObject defaultMethodObject = new InObject(con.getName(), "", parameters, true);
    defaultMethodObject.setMethod("prepareStatement");
    addToMethodList(defaultMethodObject);
  }

  public ResultSet executeQuery() throws SQLException {
    checkForClosed();
        InObject inObject = new InObject(objectName, "executeQuery", true);
    addToMethodList(inObject);
    ResultSet rset = (ResultSet)makeCall();
    return rset;
  }

  public int executeUpdate() throws SQLException {
    checkForClosed();
        InObject inObject = new InObject(objectName, "executeUpdate", true);
    addToMethodList(inObject);
    Integer i = (Integer)makeCall();
    return i.intValue();
  }

  public boolean execute() throws SQLException {
    checkForClosed();
        InObject inObject = new InObject(objectName, "execute", true);
    addToMethodList(inObject);
  Boolean b = (Boolean)makeCall();
  return b.booleanValue();
 }

 public void setNull(int parameterIndex, int sqlType) throws SQLException {
  InObject inObject = new InObject(objectName, "setNull", new Object[] {new
Integer(parameterIndex), new Integer(sqlType)}, false);
  addToMethodList(inObject);
 }

 public void setBoolean(int parameterIndex, boolean x) throws SQLException {
  InObject inObject = new InObject(objectName, "setBoolean", new Object[] {new
Integer(parameterIndex), new Boolean(x)}, false );
  addToMethodList(inObject);
 }

 public void setByte(int parameterIndex, byte x) throws SQLException {
  InObject inObject = new InObject(objectName, "setByte", new Object[] {new
Integer(parameterIndex), new Byte(x)}, false );
  addToMethodList(inObject);
 }

 public void setShort(int parameterIndex, short x) throws SQLException {
  InObject inObject = new InObject(objectName, "setShort", new Object[] {new
Integer(parameterIndex), new Short(x)}, false );
  addToMethodList(inObject);
 }

 public void setInt(int parameterIndex, int x) throws SQLException {
  InObject inObject = new InObject(objectName, "setInt", new Object[] {new
Integer(parameterIndex), new Integer(x)}, false );
  addToMethodList(inObject);
 }

 public void setLong(int parameterIndex, long x) throws SQLException {
  InObject inObject = new InObject(objectName, "setLong", new Object[] {new
Integer(parameterIndex), new Long(x)}, false );
  addToMethodList(inObject);
 }

 public void setFloat(int parameterIndex, float x) throws SQLException {
  InObject inObject = new InObject(objectName, "setFloat", new Object[] {new
Integer(parameterIndex), new Float(x)}, false );
  addToMethodList(inObject);
 }
```

```
  public void setDouble(int parameterIndex, double x) throws SQLException {
    InObject inObject = new InObject(objectName, "setDouble", new Object[] {new
Integer(parameterIndex), new Double(x)}, false );
    addToMethodList(inObject);
  }

  public void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException {
    InObject inObject = new InObject(objectName, "setBigDecimal", new Object[] {new
Integer(parameterIndex), x}, false );
    addToMethodList(inObject);
  }

  public void setString(int parameterIndex, String x) throws SQLException {
    // To be compatible with earlier drivers. By specs we should use setNull()
    if (x == null){
      setNull(parameterIndex, Types.VARCHAR);
      return;
    }
        InObject inObject = new InObject(objectName, "setString", new Object[] {new
Integer(parameterIndex), x}, false );
    addToMethodList(inObject);
  }

  public void setBytes(int parameterIndex, byte x[]) throws SQLException {
    InObject inObject = new InObject(objectName, "setBytes", new Object[] {new
Integer(parameterIndex), x}, false );
    addToMethodList(inObject);
  }

  public void setDate(int parameterIndex, java.sql.Date x) throws SQLException {
    InObject inObject = new InObject(objectName, "setDate", new Object[] {new
Integer(parameterIndex), x}, false );
    addToMethodList(inObject);
  }

  public void setTime(int parameterIndex, java.sql.Time x) throws SQLException {
    InObject inObject = new InObject(objectName, "setTime", new Object[] {new
Integer(parameterIndex), x}, false );
    addToMethodList(inObject);
  }

  public void setTimestamp(int parameterIndex, java.sql.Timestamp x) throws
SQLException {
    InObject inObject = new InObject(objectName, "setTimestamp", new Object[] {new
Integer(parameterIndex), x}, false );
    addToMethodList(inObject);
```

```
}

  public void setAsciiStream(int parameterIndex, InputStream input, int length) throws
SQLException{
      int i = 0;
      byte bytes[];
      try {
         bytes = new byte[input.available()];
         input.read(bytes);
         input.close();
      }catch(IOException ioe){
         throw new SQLException("Unable to set the Ascii stream to the statement");
      }
      InObject inObject = new InObject(objectName, "setAsciiStream", new Object[] {
new Integer(parameterIndex), bytes, new Integer(length)}, false);
      addToMethodList(inObject);
   }

   public void setUnicodeStream(int parameterIndex, InputStream input, int
length)throws SQLException{
      int i = 0;
      byte bytes[];
      try{
         bytes = new byte[input.available()];
         input.read(bytes);
         input.close();
      }catch(IOException ioe){
         throw new SQLException("Unable to set the Unicode stream to the statement");
      }
      InObject inObject = new InObject(objectName, "setUnicodeStream", new Object[]
{new Integer(parameterIndex), bytes, new Integer(length)}, false);
      addToMethodList(inObject);
   }


  public void setBinaryStream(int parameterIndex, java.io.InputStream input, int length)
throws SQLException {
   int i=0;
   int singleByte;
   byte[] bytes;
   try{
     bytes = new byte[input.available()];
     input.read(bytes);
     input.close();
   } catch(IOException ioe){
     throw new SQLException("Cannot set the binary stream to the statement");
```

```
    }
    InObject inObject = new InObject(objectName, "setBinaryStream", new Object[] {new
Integer(parameterIndex), bytes, new Integer(length)}, false);
    addToMethodList(inObject);
  }

  public void clearParameters() throws SQLException {
    session.call(objectName, "clearParameters", false);
  }

  public void setObject(int parameterIndex, Object x, int targetSqlType, int scale)throws
SQLException {
    Object[] params = new Object[] {new Integer(parameterIndex), x, new
Integer(targetSqlType), new Integer(scale) };
    Class[] paramTypes = new Class[] {Integer.class, Object.class, Integer.class,
Integer.class};
    InObject inObject = new InObject(objectName, "setObject", params, paramTypes,
false);
    addToMethodList(inObject);
  }

  public void setObject(int parameterIndex, Object x, int targetSqlType) throws
SQLException {
    Object[] params = new Object[] {new Integer(parameterIndex), x, new
Integer(targetSqlType)};
    Class[] paramTypes = new Class[] {Integer.class, Object.class, Integer.class};
    InObject inObject = new InObject(objectName, "setObject", params, paramTypes,
false);
    addToMethodList(inObject);
  }

  public void setObject(int parameterIndex, Object x) throws SQLException {
          Object[] params = new Object[] {new Integer(parameterIndex), x};
    Class[] paramTypes = new Class[] {Integer.class, Object.class};
    InObject inObject = new InObject(objectName, "setObject", params, paramTypes,
false );
    addToMethodList(inObject);
  }

  // *************** JDBC 2.0 *************

  public void addBatch() throws SQLException {
    addToMethodList(new InObject(objectName, "addBatch", true));
  }
```

```
public void setCharacterStream(int parameterIndex,
                    java.io.Reader reader,
                    int length ) throws SQLException {

}


public void setRef (int i, Ref x) throws SQLException {

}


public void setBlob (int i, Blob x) throws SQLException {

}


public void setClob (int i, Clob x ) throws SQLException {

}


public void setArray (int i, Array x ) throws SQLException {

}


public ResultSetMetaData getMetaData() throws SQLException {
  return null;
}


public void setDate(int parameterIndex, java.sql.Date x, Calendar cal) throws
SQLException {

}


public void setTime(int parameterIndex, java.sql.Time x, Calendar cal)
        throws SQLException {

}


public void setTimestamp(int parameterIndex, java.sql.Timestamp x, Calendar cal)
        throws SQLException {
  }

  public void setNull (int paramIndex, int sqlType, String typeName) throws
SQLException {

  }
 }
```

Class HTPJDBCSession

```
package com.VSDV.HTTPJDBC.Client;

import java.io.*;
import java.sql.*;
import java.net.*;

import javax.swing.Timer;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import com.VSDV.HTTPJDBC.Common.*;

public class HTTPJDBCSession {

 URL url, streamURL;
 Timer timer;
 private String sessionID;
 boolean secure;

 public HTTPJDBCSession(String host, int port, boolean secure) {
  try {
    this.secure = secure;
    String secureStr = "http";
    if (secure)
      secureStr = "https";
    else
      secureStr = "http";
         url = new URL(secureStr, host, port, "/JDBCServlet");
          createSession();
  } catch (Exception ex) {
    ex.printStackTrace();
  }
 }

 public String getID(){
  return sessionID;
 }

 public void createSession() throws Exception {
  Hashtable hash = new Hashtable();
  hash.put("action", "createSession");
  InputStream input = doGet(url, hash);
        ObjectInputStream objInput = new ObjectInputStream(input);
  sessionID = (String)objInput.readObject();
  objInput.close();
 }

 public InputStream doGet(URL u, Hashtable hash) throws Exception {
         String parameters = doGetParameters(hash);
  URL url = new URL(u.toString()+"?"+parameters);
  URLConnection urlConnection = url.openConnection();
  urlConnection.setDoInput(true);
  return urlConnection.getInputStream();
 }

 private String doGetParameters(Hashtable hash) {
         String returnStr = "";
  Enumeration enum = hash.keys();
  while (enum.hasMoreElements() ) {
          String key = (String)enum.nextElement();
          String value = (String)hash.get(key);
   returnStr += key+"="+value+"&";
         }
         return returnStr;
 }

 public InputStream doPost(URL url, byte[] data) throws IOException  {
  URLConnection urlConnection = url.openConnection();
        urlConnection.setUseCaches(false);
  urlConnection.setRequestProperty("content-length", String.valueOf(data.length ) );
  urlConnection.setDoOutput(true);
  OutputStream out = new BufferedOutputStream(urlConnection.getOutputStream() );
  for (int x=0; x < data.length; x++) {
          out.write(data[x]);
         }
  out.close();
  return new BufferedInputStream(urlConnection.getInputStream() );
 }


 public Object call(String objName, String method, boolean hasReturnValue) throws
SQLException {
  return call(objName, method, null,hasReturnValue);
 }

 public Object call(String objName, String method, Object[] params, Class[]
paramTypes, boolean hasReturnValue) throws SQLException {
  InObject inObj = new InObject(sessionID, objName, method, params,
paramTypes,hasReturnValue);
```

```
  return makeCall(inObj);
 }

 public Object call(String objName, String method, Object[] params, boolean
hasReturnValue) throws SQLException {
        InObject inObj = new InObject(sessionID, objName, method, params,
hasReturnValue);
  return makeCall(inObj);
 }

 private Object makeCall(InObject inObject) throws SQLException {
  try {
                ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
                ObjectOutputStream objOut = new ObjectOutputStream(byteOut);
                 objOut.writeObject(inObject);
                 byte[] paramData = byteOut.toByteArray();
   InputStream in=null;
   if (inObject.hasReturnValue( ) ){
    in = doPost(url, paramData);
    ObjectInputStream objIn = new ObjectInputStream(in);
    Object readObject = (Object)objIn.readObject();
    objIn.close();
                 if (readObject instanceof NullObject) return null;
    if (readObject instanceof SQLException) {
                   throw (SQLException)readObject;
          } else if (readObject instanceof Exception) {
                   throw new SQLException( ((Exception)readObject).getMessage() );
    } else {
                   return readObject;
    }
   } else {
    doPost(url, paramData);
    return null;
   }
      } catch (Exception ex) {
                 ex.printStackTrace();
                 throw new SQLException( ex.getMessage() );
        }
 }

 public Class[] getParameterTypes(Object[] params) {
  if (params == null) return null;
  Class[] paramTypes = new Class[params.length];
        for (int x=0; x < paramTypes.length; x++) {
   Class c =  params[x].getClass();
   paramTypes[x] = adjustIt(c);
```

```
        }
        return paramTypes;
    }

    private Class adjustIt(Class c) {
     if (Integer.class.equals(c) ) return Integer.TYPE;
     else if (Boolean.class.equals(c) ) return Boolean.TYPE;
     else if (Character.class.equals(c) ) return Character.TYPE;
     else if (Byte.class.equals(c) ) return Byte.TYPE;
     else if (Short.class.equals(c) ) return Short.TYPE;
     else if (Long.class.equals(c) ) return Long.TYPE;
     else if (Float.class.equals(c) ) return Float.TYPE;
     else if (Double.class.equals(c) ) return Double.TYPE;
     else return c;
    }

    public Object call(ArrayList methodList) throws SQLException{
     OutObject outObject = null;
     try {
               ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
               ObjectOutputStream objOut = new ObjectOutputStream(byteOut);
                objOut.writeObject(methodList);
                byte[] paramData = byteOut.toByteArray();
      InputStream in=null;
      in = doPost(url, paramData);
      ObjectInputStream objIn = new ObjectInputStream(in);
      Object returnObject = objIn.readObject();
      objIn.close();
                if (returnObject instanceof NullObject){
       return null;
      }
      if( returnObject instanceof OutObject){
       outObject = (OutObject)returnObject;
      }
      if (returnObject instanceof SQLException) {
            throw (SQLException)returnObject;
      } else if (returnObject instanceof Exception) {
                throw new SQLException(((Exception)returnObject).getMessage());
      } else {
                return outObject;
      }
     } catch (Exception ex) {
                ex.printStackTrace();
                throw new SQLException( ex.getMessage() );
        }
    }
```

```
}
```

Class HTTPJDBCStatement

```
package com.VSDV.HTTPJDBC.Client;

import java.sql.*;
import java.util.*;

import com.VSDV.HTTPJDBC.Common.*;

/**
 * Implements remoted functionality for java.sql.Statement
 */
public class HTTPJDBCStatement extends JDBCStub implements Statement {

  protected HTTPJDBCConnection con;
  protected HTTPJDBCSession session;
  protected MethodList methodList = new MethodList();
  protected boolean closed = false;
  public HTTPJDBCStatement(HTTPJDBCConnection con) {
    this.con = con;
          session = con.getHTTPJDBCSession();
    initializeStatement(con, null);
  }

  public HTTPJDBCStatement(HTTPJDBCConnection con, Object[] parameters){
    this.con = con;
          session = con.getHTTPJDBCSession();
    initializeStatement(con, parameters);
  }

  public void initializeStatement(HTTPJDBCConnection con, Object[] parameters){
    InObject defaultMethodObject = new InObject(con.getName(), "", false);
    defaultMethodObject.setMethod("createStatement");
    addToMethodList(defaultMethodObject);
  }

  public ResultSet executeQuery(String sql) throws SQLException {
    checkForClosed();
    addToMethodList(new InObject(session.getID(), objectName, "executeQuery", new
Object[] {sql}, true));
    ResultSet rset = (ResultSet)makeCall();
    return rset;
  }

  public ResultSet executeQueryAndClose(String sql) throws SQLException{
```

```
  checkForClosed();
  addToMethodList(new InObject(session.getID(), objectName, "executeQuery", new
Object[] {sql}, true));
  InObject inObject = new InObject(objectName, "close", false);
  addToMethodList(inObject);
  ResultSet rset = (ResultSet)makeCall();
  methodList.removeAll(methodList);
  closed = true;
  return rset;
 }

 public void close() throws SQLException {
  session.call(objectName, "close", false);
  closed = true;
 }

 public boolean executeAndClose(String sql) throws SQLException {
  checkForClosed();
  InObject inObject = new InObject(objectName, "execute", new Object[] {sql}, true );
  addToMethodList(inObject);
  Boolean b = (Boolean)makeCall();
  return b.booleanValue();
 }

 public boolean execute(String sql) throws SQLException {
  checkForClosed();
  InObject inObject = new InObject(objectName, "execute", new Object[] {sql}, true );
  addToMethodList(inObject);
  Boolean b = (Boolean)makeCall();
  return b.booleanValue();
 }

 protected void checkForClosed() throws SQLException{
  if (closed){
   throw new SQLException("Error : called method on a closed Statement");
  }
 }

 protected Object makeCall() throws SQLException{
  OutObject resultObject = (OutObject)session.call(methodList);
  tagWithReferenceNameAndRefresh(resultObject.getName());
  return resultObject.getResultObject();
 }

 private void tagWithReferenceNameAndRefresh(String name){
  setName(name);
  methodList.setName(name);
  methodList.removeAll(methodList);
 }

 public int executeUpdate(String sql) throws SQLException {
  checkForClosed();
  addToMethodList(new InObject(session.getID(), objectName, "executeUpdate", new
Object[] {sql}, true));
  Integer i = (Integer)makeCall();
       return i.intValue();
 }

 public void setEscapeProcessing(boolean enable) throws SQLException {
       session.call(objectName, "setEscapeProcessing", new Object[] {new
Boolean(enable)}, false );
 }

 public int getMaxFieldSize() throws SQLException {
       Integer i = (Integer)session.call(objectName, "getMaxFieldSize", true);
       return i.intValue();
 }

 public void setMaxFieldSize(int max) throws SQLException {
  session.call(objectName, "setMaxFieldSize", new Object[] {new Integer(max)}, false);
 }

 public int getMaxRows() throws SQLException {
  Integer i = (Integer)session.call(objectName, "getMaxRows", true );
       return i.intValue();
 }

 public void setMaxRows(int max) throws SQLException {
  addToMethodList(new InObject(session.getID(), objectName, "setMaxRows", new
Object[] {new Integer(max)}, true));
 }

 public int getQueryTimeout()throws SQLException {
  Integer i = (Integer)session.call(objectName, "getQueryTimeout",true);
       return i.intValue();
 }

 public void setQueryTimeout(int seconds) throws SQLException {
       session.call(objectName, "setQueryTimeout", new Object[] {new
Integer(seconds)}, false );
 }
```

```
public void cancel() throws SQLException {
        session.call(objectName, "cancel", false );
}

public SQLWarning getWarnings() throws SQLException {
        return null;
}
public void clearWarnings() throws SQLException {

}

public void setCursorName(String name) throws SQLException {
  session.call(objectName, "setCursorName", new Object[] {name}, false );
}

public ResultSet getResultSet() throws SQLException {
  return null;
}

public int getUpdateCount() throws SQLException {
        Integer i = (Integer)session.call(objectName, "getUpdateCount", true);
        return i.intValue();
}

public boolean getMoreResults() throws SQLException {
  Boolean i = (Boolean)session.call(objectName, "getMoreResults", true );
        return i.booleanValue();
}

public void setFetchDirection(int direction) throws SQLException {

}

public int getFetchDirection() throws SQLException {
  return 1;
}

public void setFetchSize(int rows) throws SQLException {
  addToMethodList(new InObject(session.getID(), objectName, "setFetchSize", new
Object[] {new Integer(rows)}, true));
}

public int getFetchSize() throws SQLException {
  return 1;
}
```

```
public int getResultSetConcurrency() throws SQLException {
        return 1;
}

public int getResultSetType() throws SQLException {
        return 1;
}

public void addBatch(String sql) throws SQLException {
  addToMethodList(new InObject(objectName, "addBatch", new Object[] {sql}, true));
}

public void clearBatch() throws SQLException {

}

public int[] executeBatch() throws SQLException {
  addToMethodList(new InObject(objectName, "executeBatch", true));
  Object obj = makeCall();
  return new int[] {1, 2};
}

public Connection getConnection() throws SQLException {
        return con;
}

HTTPJDBCConnection getHTTPJDBCConnection() {
  return con;
}

protected void addToMethodList(InObject object){
  object.setSessionID(session.getID());
  methodList.add(object);
}

public void finalize(){
  if (!closed){
    try{
      close();
    } catch(SQLException sqle){
      sqle.printStackTrace();
    }
  }
}
}
```

Class JDBCStub

```
package com.VSDV.HTTPJDBC.Client;

import java.sql.*;

/**
 * Simple root class to be used for stuff like HTTPJDBCStatement etc.
 * Basically forces the child objects to use an object name
 */

public abstract class JDBCStub {

  public String objectName=null;

  public JDBCStub(String objectName) {
        this.objectName = objectName;
  }

  public JDBCStub(){
  }

  public void setName(String name) {
        objectName = name;
  }

  public String getName(){
   return objectName;
  }
}
```

Common Classes Used by both the Server and Client

Class InObject

```
package com.VSDV.HTTPJDBC.Common;

import java.io.*;


/**
 * Object that is serialized containing information about which
 * object to call which method on with the method's parameters
 * It also contains the sessionId used by the server to allow
 * for sessions over HTTP
 * Cookies were originally used then abandoned because
 * the Java plugin did not seem to retrieve cookie data
 * from the servlet hence not passing it back upon subsequent
 * calls to the servlet
 */

/*
 To Do :
 1. Make this object much lighter.
 2. Let the servlet do the createInputStreamFromBytes
*/
public class InObject implements Serializable{

 static final long serialVersionUID = 4064151811232982612L;
 public String method, objectName;
 public Object[] parameters;
 public Class[] parameterTypes = null;
 boolean returnsStream = false;
 String sessionId;
 boolean hasReturnValue;    // to avoid null return

 public InObject(String sessionId, String objectName, String method, Object[]
parameters, boolean hasReturnValue) {  // to avoid null return
        this.sessionId = sessionId;
        this.objectName = objectName;
  this.hasReturnValue = hasReturnValue;
  this.method = method;
  this.parameters = parameters;
  if (parameters != null)
   this.parameterTypes = getTypes(parameters);
  else
```

```
    this.parameterTypes = null;
  }

  public InObject(String sessionId, String objectName, String method,
                Object[] parameters, Class[] parameterTypes,
                boolean hasReturnValue ) {
    this(sessionId, objectName, method, parameters, hasReturnValue);
    this.parameterTypes = parameterTypes;
  }

  public InObject(String objectName, String method, boolean hasReturnValue){
    this(null, objectName, method, null, hasReturnValue);
  }

  public InObject(String objectName, String method, Object[] parameters, boolean
hasReturnValue){
    this(null, objectName, method, parameters, hasReturnValue);
  }

  public InObject(String objectName, String method, Object[] parameters, Class[]
parameterTypes, boolean hasReturnValue ) {
    this(null, objectName, method, parameters, parameterTypes, hasReturnValue);
  }

  public String getSessionId() { return sessionId; }

  public String getMethodName() { return method; }

  public String getObjectName() { return objectName; }

  public Object[] getParameters() {
    return parameters;
  }

 /* public void createInputStreamFromBytes(){
    if (!getMethodName().equals("setBinaryStream")){
      return;
    }
    if ( !(parameters[1] instanceof ByteArrayInputStream)){
      parameters[1] = getInputStreamFromBytes((byte[])parameters[1]);
      parameterTypes[1] = java.io.InputStream.class;
    }
  }*/
  public void createInputStreamFromBytes(){
   if(getMethodName().equals("setBinaryStream") ||
getMethodName().equals("setAsciiStream") ||
getMethodName().equals("setUnicodeStream")){
     System.out.println("getMethodName : " + getMethodName());
     if(!(parameters[1] instanceof ByteArrayInputStream)){
       parameters[1] = getInputStreamFromBytes((byte[])parameters[1]);
       parameterTypes[1] = java.io.InputStream.class;
     }
   }
  }


  public Class[] getParameterTypes() {
        return adjustThem(parameterTypes);
  }

  public boolean hasReturnValue ( ){
   return  hasReturnValue ;
  }

  private Class[] getTypes(Object[] params) {
        if (params == null) return null;
    Class[] classes = new Class[params.length];
    for (int x=0; x < params.length; x++) {
      classes[x] = params[x].getClass();
    }
          return classes;
  }

  private Class[] adjustThem(Class[] classes) {
          if (classes == null) return null;
    for (int x=0; x < classes.length; x++) {
      classes[x] = adjustIt(classes[x]);
          }
          return classes;
  }

  private Class adjustIt(Class c) {
    if (Integer.class.equals(c) ) return Integer.TYPE;
    else if (Boolean.class.equals(c) ) return Boolean.TYPE;
    else if (Character.class.equals(c) ) return Character.TYPE;
    else if (Byte.class.equals(c) ) return Byte.TYPE;
    else if (Short.class.equals(c) ) return Short.TYPE;
    else if (Long.class.equals(c) ) return Long.TYPE;
    else if (Float.class.equals(c) ) return Float.TYPE;
    else if (Double.class.equals(c) ) return Double.TYPE;
  else return c;
 }

 public boolean getReturnsStream() { return returnsStream; }

 public String toString(){
  return "Session : "+getSessionId()+" Object Name : "+getObjectName()+"
"+getMethodName();
 }

 public void setSessionID(String sessionID){
  this.sessionId = sessionID;
 }

 public void setMethod(String method){
  this.method = method;
 }

 private InputStream getInputStreamFromBytes(byte[] bytes){
  return new ByteArrayInputStream(bytes);
 }
}
```

Class MethodList

```
package com.VSDV.HTTPJDBC.Common;

import java.util.*;

public class MethodList extends ArrayList{
  private String listName;
  public MethodList(){
  }

  public void setName(String name){
    listName = name;
  }

  public String getName(){
    return listName;
  }
}
```

Class NullObject

```
package com.VSDV.HTTPJDBC.Common;

import java.io.*;


/**
 * Null object used for remote method calls because passing null around doesn't work
 * it needs it's own object
 * The serialVersionUID is to insure this class can be recompiled
 * and still be passed around without the remote end choking
 * because the serialVersionUID are different.
 */
public class NullObject implements Serializable {

  static final long serialVersionUID = 1333822856806054836L;

}
```

Class OutObject

```
package com.VSDV.HTTPJDBC.Common;
import java.io.*;

public class OutObject implements Serializable{
  private String name;
  private Object result;
  public OutObject(String name, Object result) {
    this.name = name;
    this.result = result;
  }

  public String getName(){
    return name;
  }

  // Result Object can be instanceof ResultSet, SQLException, Exception etc.
  public Object getResultObject(){
    return result;
  }
}
```

Server Side Classes

Class DBConnectionFactory

```
package com.VSDV.HTTPJDBC.Server;

import java.sql.*;
import java.util.*;

public class DBConnectionFactory {
  private DBConnectionFactory() {
  }

  public static Connection createConnection(Properties properties) throws
SQLException{
    String vendor = (String)properties.get("DBVENDOR");
    DBVendor dbVendor = null;
    if (vendor.equals("ORACLE")){
      dbVendor = new Oracle(properties);
    } else if (vendor.equals("SQLSERVER")){
      dbVendor = new SQLServer(properties);
    }
    if (dbVendor == null){
      throw new SQLException("Fatal error - No database vendor specified");
    }
    return dbVendor.createConnectionFromProperties();
  }
}
```

Class DBVendor

```
package com.VSDV.HTTPJDBC.Server;

import java.sql.*;

public interface DBVendor {
  public Connection createConnectionFromProperties() throws SQLException;
}
```

Class JDBCServerConnection

```
package com.VSDV.HTTPJDBC.Server;

import com.VSDV.VerticalSuite.*;
import javax.servlet.http.*;
import javax.servlet.*;
import java.sql.*;
import java.io.*;

public class JDBCServerConnection implements HttpSessionBindingListener{
  private Connection connection;
  private String connectionName;
  private ServletContext context;
  public JDBCServerConnection(String name, Connection connection, ServletContext context) {
    this.connection = connection;
    connectionName = name;
    this.context = context;
  }

  public void valueBound(HttpSessionBindingEvent e){
    HttpSession session = e.getSession();
    FileOutputStream file = (FileOutputStream)session.getAttribute(GenericLoginServlet.VSLOGFILE);
    try{
      if (file != null){
        String s = "Connection Made : "+connectionName;
        file.write(s.getBytes());
      }
    }catch(IOException ioe){
        ioe.printStackTrace();
    }
  }

  // The following method does the following
  // when the session is invalidated/timed out
  //  1. db connection is closed.
  //  2. removes this object from the session application layers data.
  public void valueUnbound(HttpSessionBindingEvent e){
    HttpSession session = e.getSession();
    context.log("Unbound : "+connectionName);
    try{
      connection.close();
    } catch(SQLException sqle){
      try {
        FileOutputStream file = (FileOutputStream)session.getAttribute(GenericLoginServlet.VSLOGFILE);
        if (file != null){
          String s = "********** Connection : "+connectionName+" Closed. ********";
          file.write(s.getBytes());
          sqle.printStackTrace(new PrintStream(file));
        }
      }catch(IOException ioe){
        ioe.printStackTrace();
        context.log(ioe.getMessage());
      }
    }
    // remove this from the current session
    ServerSessions sessions = JDBCServlet.getServerSessions();
    sessions.disposeSession(connectionName);
    session.removeAttribute(connectionName);
  }
}
```

Class JDBCServerSession

```
package com.VSDV.HTTPJDBC.Server;

import java.io.*;
import java.util.*;
import javax.servlet.http.*;
import javax.servlet.*;
import java.sql.*;
import java.lang.reflect.*;



/**
 * Session used by servlet corresponding to LFRT Applet
 * client user loging session
 * Holds JDBC objects on behalf of the user
 */
public class JDBCServerSession implements Serializable {

  ServletContext context;
  Driver driver;
  Hashtable hash;
  int objNameCounter=1;
  long lastRenewed;
  String sessionId;
  Vector objectNameIndex = new Vector();

  public JDBCServerSession(ServletContext context) {
   this.context = context;
   hash = new Hashtable();
  }

  public void setSessionId(String id) {
   sessionId = id;
  }

  public String getSessionId() {
        return sessionId;
  }

  public long getLastRenewed() {
   return lastRenewed;
  }
 /**
  * Returns objects from session based on a name
  */
 public Object getObject(String objectName) {
  return hash.get(objectName);
 }

 public Object[] getObjects() {
  Object[] objects = new Object[hash.size()];
          Enumeration enum = hash.elements();
          int count = 0;
          while (enum.hasMoreElements() ) {
            objects[count] = (Object)enum.nextElement();
            count++;
  }
          return objects;
 }

 /**
  * Sets objects into the session based on a name
  */
 public String setObject(Object obj) {
          String objectName = objNameCounter+"";
  hash.put(objectName, obj);
//   context.log(objectName+" setObject = "+ obj);
  objNameCounter++;
  objectNameIndex.addElement(objectName);
  return objectName;
 }

 /**
  * Renews keepalive or lease for this session so it
  * doesn't get killed
  */
 public void renew() {
  lastRenewed = System.currentTimeMillis();
 }

 /**
  * Closes up all objects associated with this session
  */
 public void close() {
  try {
    Enumeration enum = hash.elements();
    while (enum.hasMoreElements() ) {
      Object obj = enum.nextElement();
```

09872502.060101

```
        closeObject(obj);
                }
            } catch (Exception ex) { ex.printStackTrace(); }
    }

    /**
     * Closes out JDBC objects when this session is tossed in the trash
     */
    public void closeObject(Object obj, String objectName) {
            try {
      Class c = obj.getClass();
      Method method = c.getMethod("close", null);
      method.invoke(obj, null);
      hash.remove(objectName);
     } catch (Exception ex) {
                context.log("Error closing object "+obj.toString());
            }
    }

    public void closeObject(Object obj) {
            try {
      Class c = obj.getClass();
      Method method = c.getMethod("close", null);
      method.invoke(obj, null);
      hash.remove(obj);
     } catch (Exception ex) {
                context.log("Error closing object "+obj.toString());
            }
    }

}
```

Class JDBCServlet

```
package com.VSDV.HTTPJDBC.Server;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import java.sql.*;
import com.VSDV.RowSet.*;

import com.VSDV.HTTPJDBC.Common.*;

public class JDBCServlet extends HttpServlet{

  public static ServerSessions sessions;
  Driver poolDriver;
  Properties poolProps;
  ServletContext context;
  ServletConfig config;

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
          context = config.getServletContext();
          sessions = new ServerSessions(context);
    try {
      this.config = config;
      poolDriver = (Driver) Class.forName("weblogic.jdbc20.oci.Driver").newInstance();
    } catch (Exception ex) { context.log("", ex); }
  }

  public static ServerSessions getServerSessions() {
        return sessions;
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
//   HttpSession session = req.getSession();
    String action = req.getParameterValues("action")[0];
    if (action.equals("renew") ) {
      String sessionId = req.getParameterValues("sessionid")[0];
      JDBCServerSession s = sessions.getSession(sessionId);
      s.renew();
          } else if (action.equals("createSession") ) {
```

```
    String sessionId = sessions.addSession(new JDBCServerSession(context));
    OutputStream out = res.getOutputStream();
    ObjectOutputStream objOut = new ObjectOutputStream(out);
    objOut.writeObject(sessionId);
    objOut.flush();
    objOut.close();
//     session.setAttribute(sessionId, new Session(sessionId, context));
        }
  }

  private void sendResultObject(HttpServletResponse res, Object resultObject) throws
IOException{
   OutputStream out = res.getOutputStream();
        ObjectOutputStream objOut = new ObjectOutputStream(out);
   objOut.writeObject(resultObject);
   objOut.flush();
   objOut.close();
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
//   HttpSession session = req.getSession();
   Object returnValue=null, obj=null;
        context = config.getServletContext();
   Object inObject = null;
   try {
    InputStream in = req.getInputStream();
    ObjectInputStream objIn = new ObjectInputStream(in);
    inObject = objIn.readObject();
    objIn.close();
    if (inObject instanceof InObject){
      returnValue = executeCall((InObject)inObject, req);
    } else if(inObject instanceof MethodList){
      returnValue = executeMethods((MethodList)inObject, req);
    }
   } catch (InvocationTargetException ite) {
         returnValue = ite.getTargetException();
         context.log("InvocationTargetException", ite);
   } catch (RuntimeException re) {
    context.log("Null Pointer Exception", re);
   } catch (Exception ex) {
         returnValue = ex;
         context.log("Exception", ex);
        }
   if (returnValue == null){
    sendResultObject(res, new NullObject());
   } else{
     try{
       sendResultObject(res, getReturnValue(returnValue));
     } catch(SQLException sqle){
       sendResultObject(res, sqle);
     }
   }
 }

 private String createConnection(InObject callObject, HttpServletRequest req) throws
SQLException, InvocationTargetException, NoSuchMethodException,
IllegalAccessException{
   Properties props = (Properties)callObject.parameters[1];
   java.sql.Connection con = DBConnectionFactory.createConnection(props);
   String connectionName = createObjectReference(callObject, con).toString();
//   setupConnectionTimeout(req, con, callObject.getSessionId());   // comment this line
for local testing
   log("Connection Created");
   return connectionName;
 }

 private void setupConnectionTimeout(HttpServletRequest req, java.sql.Connection
connection, String sessionId){
   log("************************************** Setting up Connection Timeout
**********************");
   HttpSession session = req.getSession();
   session.setAttribute(sessionId,new JDBCServerConnection(sessionId, connection,
getServletContext()));
 }


 private Object executeCall(InObject callObject, HttpServletRequest req) throws
SQLException, InvocationTargetException, NoSuchMethodException,
IllegalAccessException{
   Object obj = null, returnValue = null;
   JDBCServerSession serverSession =
(JDBCServerSession)sessions.getSession(callObject.getSessionId() );
   String objectName = callObject.getObjectName();
   if (objectName.equals("driver") ) {
     returnValue = createConnection(callObject, req);  // Currently this is called when
driver.connect() is called Need to implement other methods getMinorVersion etc. if
required.
   } else if (objectName.equals("connection") ) {
     obj = serverSession.getObject("connection");
     Object methodCallReturnObject = executeMethodOnObject(obj, callObject);
     returnValue = createObjectReference(callObject, methodCallReturnObject);
```

```
  } else if (serverSession != null) {
    obj = serverSession.getObject(objectName);
    Object methodCallReturnObject = executeMethodOnObject(obj, callObject);
    returnValue = createObjectReference(callObject, methodCallReturnObject);
  }
  if (returnValue == null){
    return new NullObject();
  }
  return returnValue;
}

public Object executeMethods(MethodList methodList, HttpServletRequest req) throws
SQLException, InvocationTargetException, IllegalAccessException,
NoSuchMethodException{
  Iterator iterator = methodList.iterator();
  Object returnValue = null;
  Object objectToExecuteMethodsOn = null;
  String objectName = methodList.getName();
  while (iterator.hasNext()){
    InObject inObject = (InObject)iterator.next();
    try{
      if (objectName == null){
        objectName = (String)executeCall(inObject, req);
        methodList.setName(objectName);
      } else{
        inObject.objectName = objectName;
        objectToExecuteMethodsOn = getObjectByName(inObject);
        Object methodReturnValue =
executeMethodOnObject(objectToExecuteMethodsOn, inObject);
        if (methodReturnValue != null){
          returnValue = getReturnValue(methodReturnValue);
        }
      }
    } catch(NoSuchMethodException nsme){
      context.log("Exception : ",nsme);
    }
  }
  returnValue = getReturnValue(returnValue);  // if the return value is a ResultSet then
we need to create a cachedRowSet from that
  return new OutObject(objectName, returnValue);
}

private Object getReturnValue(Object returnValue) throws SQLException{
  if (returnValue == null){
    return new NullObject();
  }
  if (returnValue instanceof ResultSet) {
    CachedRowSet crs = new CachedRowSet();
    crs.populate((ResultSet)returnValue);
    ResultSet rset = (ResultSet)returnValue;
    rset.close();
    return crs;
  }
  return returnValue;
 }

 private Object executeMethodOnObject(Object object, InObject inObject) throws
NoSuchMethodException, InvocationTargetException, IllegalAccessException{
  Object returnValue = null;
  if (inObject.getMethodName().equals("close") ) {
    JDBCServerSession session = sessions.getSession(inObject.getSessionId());
          session.closeObject(object, inObject.getObjectName());
    return new NullObject();
          }
  inObject.createInputStreamFromBytes();
          Class calledObjectClass = object.getClass();
  Class[] pmtypes = inObject.getParameterTypes();
          Method method = calledObjectClass.getMethod(inObject.getMethodName(),
pmtypes );
  returnValue = method.invoke(object, inObject.getParameters());
  return returnValue;
 }

 private Object executeMethodsOnObject(Iterator inObjects, Object
objectToExecuteMethodsOn) throws NoSuchMethodException,
InvocationTargetException, IllegalAccessException{
  Object returnValue = null;
  while (inObjects.hasNext()){
    InObject inObject = (InObject)inObjects.next();
    returnValue = executeMethodOnObject(objectToExecuteMethodsOn, inObject);
  }
  return returnValue;
 }

 private Object getObjectByName(InObject inObject){
   JDBCServerSession serverSession =
(JDBCServerSession)sessions.getSession(inObject.getSessionId());
   return serverSession.getObject(inObject.objectName);
 }

  private Object createObjectReference(InObject inObject, Object referenceObject)
throws SQLException, InvocationTargetException, IllegalAccessException{
   JDBCServerSession serverSession =
(JDBCServerSession)sessions.getSession(inObject.getSessionId());
   String methodName = inObject.getMethodName();
   if ( methodName.equals("createStatement") ||
methodName.equals("prepareStatement")
       || methodName.equals("prepareCall") || methodName.equals("getMetaData")
       || methodName.equals("connect")) {
     return serverSession.setObject(referenceObject);
   }
   return referenceObject;
  }
}


class Session implements HttpSessionBindingListener{
   private String id;
   private ServletContext context;
   public Session(String id, ServletContext context){
     this.id = id;
     this.context = context;
   }

   public void valueBound(HttpSessionBindingEvent e){
     // do nothing, cos we would just create the object
     context.log("*********** Bound : "+id+"  *****************");
   }

   public void valueUnbound(HttpSessionBindingEvent e){
    // when the sesion dies, delete it from the hash table, and all the
    // objects owned by that session should theoretically die.
    // context.log("Unbound Session : "+id);
     context.log("********** Unbound : "+id+"  *****************");
     JDBCServlet.getServerSessions().disposeSession(id);
   }
}
```

Class Oracle.java

```
package com.VSDV.HTTPJDBC.Server;

import java.util.*;
import java.sql.*;

public class Oracle implements DBVendor{
 private final String dbDriver = "weblogic.jdbc20.oci.Driver";
 Properties properties;
 public Oracle(Properties properties) {
  this.properties = properties;
 }

 public Connection createConnectionFromProperties() throws SQLException{
  try{
   Driver driver = (Driver) Class.forName(dbDriver).newInstance();
   // for oracle oci connections the value of server should be one of
   // the tnsnames entries, but this will be the values of db parameter
   // so this needs to be changed and all we need
   // is user, password, server.
   Properties props = new Properties();
   props.put("user", properties.get("user"));
   props.put("password", properties.get("password"));
   props.put("server", properties.get("db"));
   Connection connection = driver.connect("jdbc20:weblogic:oracle", props);
   return connection;
  } catch(ClassNotFoundException cnfe){
   throw new SQLException("Fatal Error - Suitable driver class not found for Oracle");
  } catch(IllegalAccessException iae){
   throw new SQLException("Fatal Error - Illegal access exception while trying to load
driver for Oracle");
  } catch(InstantiationException ie){
   throw new SQLException("Fatal Error - Cannot instantiate driver for Oracle");
  }
 }
}
```

Class ServerSessions

```
package com.VSDV.HTTPJDBC.Server;

import java.util.*;
import java.awt.event.*;
import javax.swing.Timer;
import javax.servlet.*;


/**
 * Holds JDBCServerSession objects for the servlet
 */
public class ServerSessions {

  Hashtable hash;

  int counter=0;
  long time = System.currentTimeMillis();
  int secondsTillKill = 30;
  Timer timer;
  ServletContext context;

  public ServerSessions(ServletContext context) {
          this.context = context;
    hash = new Hashtable();
// diabling sessions
//  timer = new Timer(1000*10, new TimerListener() );
//        timer.start();
  }


  /**
   * Periodically runs and cleans up session that
   * have not been renewed lately
   */
  public class TimerListener implements ActionListener {
          public void actionPerformed(ActionEvent evt) {
            try {
      disposeOldSessions();

      } catch (Exception ex) {
              ex.printStackTrace();
            }
          }
}


public JDBCServerSession[] getSessions() {
  JDBCServerSession[] sessions = new JDBCServerSession[hash.size()];
  Enumeration enum = hash.elements();
  int count = 0;
  while (enum.hasMoreElements() ) {
    sessions[count] = (JDBCServerSession)enum.nextElement();
    count++;
        }
        return sessions;
}

/**
 * Tosses old sessions
 */
private void disposeOldSessions() {
  Enumeration enum = hash.keys();
  while (enum.hasMoreElements() ) {
          String key = (String)enum.nextElement();

          JDBCServerSession session = (JDBCServerSession)hash.get(key);
    long lastRenewed = session.getLastRenewed();
    long currentTime = System.currentTimeMillis();
    long difference = currentTime - lastRenewed;

    if ( difference > (1000*secondsTillKill) ) {
      hash.remove(key);
      session.close();
//      context.log("removed session "+key );
  //    context.log("difference = "+ difference);
    }
        }
}

/**
 * Adds session to collection
 */
public String addSession(JDBCServerSession obj) {
  counter++;
  String sessionId = time+counter+"";
  hash.put(sessionId, obj);
  obj.setSessionId(sessionId);
  return sessionId;
}
```

09872502.060101

```
 /**
  * Get session out of collection based on the sessionId
  */
 public JDBCServerSession getSession(String sessionId) {
  return (JDBCServerSession)hash.get(sessionId);
 }

 public void disposeSession(String sessionID){
  JDBCServerSession session = getSession(sessionID);
//   context.log("Closing off objects for session : "+sessionID);
  session.close();
  hash.remove(sessionID);
 }
}
```

Class SQLServer

```
package com.VSDV.HTTPJDBC.Server;

import java.util.*;
import java.sql.*;

public class SQLServer implements DBVendor {
 private String driverClass = "weblogic.jdbc.mssqlserver4.Driver";
 private Properties properties;
 public SQLServer(Properties props) {
  this.properties = props;
 }

 public Connection createConnectionFromProperties() throws SQLException{
  try{
   Driver driver = (Driver) Class.forName(driverClass).newInstance();
   Connection connection = driver.connect("jdbc:weblogic:mssqlserver4", properties);
   return connection;
  } catch(ClassNotFoundException cnfe){
   throw new SQLException("Fatal Error - Suitable driver class not found for SQLServer");
  } catch(IllegalAccessException iae){
   throw new SQLException("Fatal Error - Illegal access exception while trying to load driver for
SQLServer");
  } catch(InstantiationException ie){
   throw new SQLException("Fatal Error - Cannot instantiate driver for SQLServer");
  }
 }
}
```